Implementation of the Galerkin/Linear Finite Elements Method in 1d

April 15, 2016

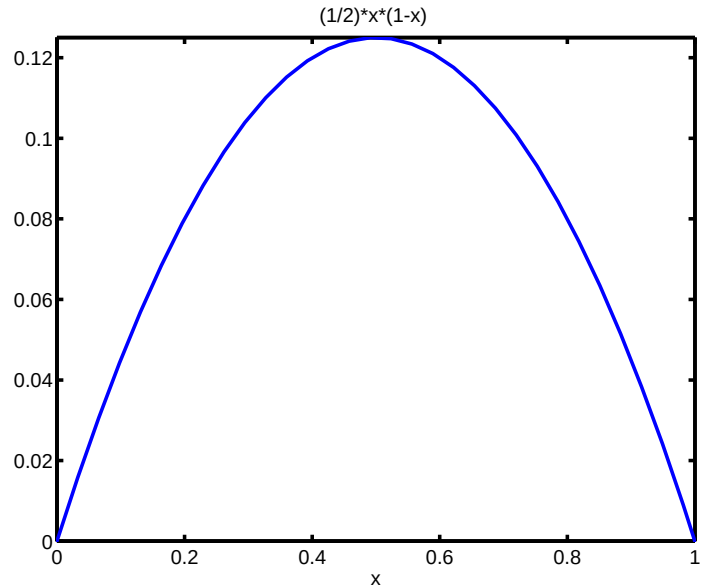
Problem Statement

We want to solve the following boundary value problem:

$$\begin{cases} -u'' = 1 & \text{in } (0,1) \\ u(0) = u(1) = 0 \end{cases} \tag{1}$$

even though we know the exact solution, which is:

$$u(x) = \frac{1}{2}x(1-x)$$



Weak Formulation

The weak form of equation (**??**) reads:

$$\begin{cases} \text{cercare } u \in V = H_0^1(0,1) t.c. \\ -\int_0^1 u'' \varphi \, dx = \int_0^1 1 \cdot \varphi \, dx \quad \forall \varphi \in V \end{cases} \tag{2}$$

Using integration by parts we get

$$-\int_0^1 u'' \varphi \, dx = \int_0^1 u' \varphi' \, dx - \varphi u' \Big|_0^1 = \int_0^1 u' \varphi' \, dx$$

Galerkin Method

Let $V_h \subset V$ be a subspace of finite dimension N_h and let $\{\varphi_i\} \subseteq V_h$ be a basis of V_h

$$\begin{cases} \text{cercare } u_h \in V_h t.c. \\ \int_0^1 u_h' \varphi_i' \, dx = \int_0^1 1 \cdot \varphi_i \, dx \quad \forall \varphi_i \in V_h \end{cases} \tag{3}$$

We can express u_h as a linear combination of basis vectors:

$$u_h = \sum_{j=1}^{N_h} u_j \varphi_j \Rightarrow u_h' = \sum_{j=1}^{N_h} u_j \varphi_j'$$

Galerkin Method

Equation **??**$_2$ becomes:

$$\sum_{j=1}^{N_h} u_j \int_0^1 \varphi_i' \varphi_j' \, dx = \int_0^1 \varphi_i \, dx \quad i = 1, \ldots, N_h$$

We therefore get a linear algebraic system of the form

$$\mathbf{Au} = \mathbf{f},$$

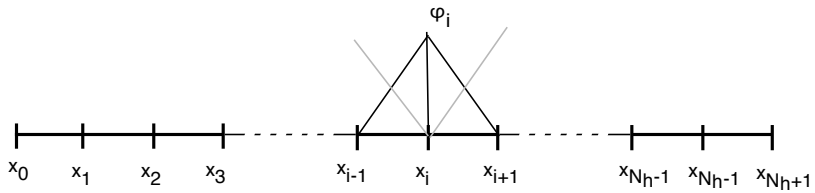
where

$$A_{ij} = \int_0^1 \varphi_i' \varphi_j' \, dx, \;\; f_i = \int_0^1 \varphi_i \, dx$$

and the vector of unknowns **u** is formed by the coefficients of the expansion of u_h w.r.t. the basis $\{\varphi_i\}$

Linear Finite Elements

Let us define a *triangulation* of the interval $(0, 1)$



Let as choose as the finite dimensional space V_h the set of functions that are continuous in $(0, 1)$ and are degree-1 polynomials (*i.e.*, affine functions) in each subintervall.
the canonical basis for V_h is gicen by:

$$\{\varphi_i\} = \{\varphi_i \in V_h \, t.c. \, \varphi_i(x_j) = \delta_{ij}\}$$

Implementation of the Linear FEM I

Using the local support property of the Finite Element basis $\varphi_i(x) \neq 0$ solo se $x \in (x_{i-1}, x_{i+1})$ we get

$$A_{ij} = \begin{cases} 0 \text{ se } |i-j| > 1 \\ \int_{x_{i-1}}^{x_i} \varphi_i'^2 + \int_{x_i}^{x_{i+1}} \varphi_i'^2 \text{ se } i = j \\ \int_{x_{i-1}}^{x_i} \varphi_i' \varphi_{i-1}' \text{ if } j = i-1 \\ \int_{x_i}^{x_{i+1}} \varphi_i' \varphi_{i+1}' \text{ if } j = i+1 \end{cases}$$

- the non zero terms in the matrix are $N_h + 2 * (N_h - 1)$ (the matrix is therefore sparse and tridiagonal)

Implementation of the Linear FEM I

Using the local support property of the Finite Element basis $\varphi_i(x) \neq 0$ solo se $x \in (x_{i-1}, x_{i+1})$ we get

$$A_{ij} = \begin{cases} 0 \text{ se } |i-j| > 1 \\ \int_{x_{i-1}}^{x_i} \varphi_i'^2 + \int_{x_i}^{x_{i+1}} \varphi_i'^2 \text{ se } i = j \\ \int_{x_{i-1}}^{x_i} \varphi_i' \varphi_{i-1}' \text{ if } j = i-1 \\ \int_{x_i}^{x_{i+1}} \varphi_i' \varphi_{i+1}' \text{ if } j = i+1 \end{cases}$$

- each entry in the matrix is given by a sum of (few) integrals each computed on only one subinterval

Implementation of the Linear FEM II

To assemble **A**, we decompose the intervals of $(0, 1)$ into integrals on subintervals:

$$A_{ij} = \int_0^1 \varphi_i' \varphi_j' \, dx = \sum_{k=1}^{N_h+1} \int_{x_{k-1}}^{x_k} \varphi_i' \varphi_j' \, dx$$

We can then use the following algorithm for assembling **A**:

1. Initialize all elements of **A** to 0
2. Compute integrals on subintervals
3. Compute entries od **A** as sums of the partial integrals

This is unnecessary for this simple case but has advantages in more complex situations we will discuss later:

1. Extension to different bases
2. Extension to multiple space dimensions
3. Parallel computing

Implementation of the Linear FEM III

In each subinterval (x_{k-1}, x_k) there are four integrals $\neq 0$ which need to be computed, they are usually arranged into a *local matrix*:

$$\mathbf{A_{loc}} = \begin{bmatrix} i = k-1, j = k-1 & i = k-1, j = k \\ i = k, j = k-1 & i = k, j = k \end{bmatrix}$$

$A_{loc_{11}}$ will then be added to $A_{k-1,k-1}$
$A_{loc_{12}}$ will then be added to $A_{k-1,k}$ etc...

this process is called assembly of the coefficient matrix

Implementation in Octave

```
1   a = 0;
2   b = 1;
3   L = b-a;
4   nnodes = 50;
5   nels   = nnodes - 1;
6   h      = L / nels;
7   nodes  = [a:h:b];
8   elements = [1:nnodes-1; 2:nnodes];
9
10  %% Initialize the coefficients matrix
11  A = zeros (nnodes);
12
13  %% loop over the intervals to assemble the coefficient matrix:
14  for iel=1:nels
15
16    %% build local matrix
17    mloc = zeros (2);
18
19    %% loop over the rows of the local matrix
20    for inode =1:2
21
22      %% compute gradient of the inode-th test function
23      igrad = (-1)^inode / h;
24
25      %% loop over the columns of the local matrix
26      for jnode =1:2
27
28        %% compute gradient of the jnode-th shape function
29        jgrad = (-1)^jnode/h;
30
31        %% integrate igrad*jgrad over the element iel and store at mloc (inode, jnode)
32        mloc(inode,jnode) = igrad * jgrad * h;
33
34      end
35    end
```

Implementation in Octave

```
30
31          %% integrate igrad*jgrad over the element iel and store at mloc (inode, jnode)
32          mloc(inode,jnode) = igrad * jgrad * h;
33
34        end
35      end
36
37      %% Assembly:
38      for inode =1:2
39        for jnode =1:2
40          A(elements(inode,iel), elements(jnode,iel)) = ...
41          A(elements(inode,iel), elements(jnode,iel)) + ...
42          mloc(inode,jnode);
43        end
44      end
45    end
46
47
48    %% Initialize the rhs vector
49     f = zeros (nnodes, 1);
50
51    %% loop over the elements to assemble the rhs
52     for iel = 1:nels
53
54      %% build local vector
55       vloc = zeros (2, 1);
56
57      %% loop over the rows of local vector
58       for inode =1:2
59
60         %% compute the integral of the forcing term
61         %% times the inode-th test function integrated
62         %% on the interval iel
63          vloc(inode) = h / 2;
64
65       end
```

Implementation in Octave

```
60        %% compute the integral of the forcing term
61        %% times the inode-th test function integrated
62        %% on the interval iel
63         vloc(inode) = h / 2;
64
65       end
66
67       %% Assembly:
68       for inode = 1:2
69         f(elements(inode, iel)) += ...
70         vloc(inode);
71       end
72     end
73     %% Impose boundary conditions via Iron's trick
74     f(1)   = 0;
75     f(end) = 0;
76
77     A(1, 1) = 1;
78     A(1, 2:end) = 0;
79
80     A(end, end) = 1;
81     A(end, 1:end-1) = 0;
82
83     %% Solve linear system
84     uh = A \ f;
85
86     %% plot solution
87     plot (nodes, uh, nodes, nodes .* (1 - nodes) / 2, 'x')
```

Implementation in C++

```
1   #include <iostream>
2   #include <cmath>
3
4   int main ()
5   {
6
7     constexpr double a = 0;
8     constexpr double b = 1;
9     constexpr double L = b - a;
10    constexpr unsigned int nnodes = 50;
11    constexpr unsigned int nels = nnodes - 1;
12    constexpr double h = L / static_cast<double> (nels);
13    constexpr unsigned int maxit = 10000;
14    constexpr double tol = 1.0e-15;
15
16    double nodes[nnodes];
17    for (unsigned int ii = 0; ii < nnodes; ++ii)
18      nodes[ii] = static_cast<double>(ii) * h + a;
19
20    unsigned int elements[nels][2];
21    for (unsigned int ii = 0; ii < nels; ++ii)
22      {
23        elements[ii][0] = ii;
24        elements[ii][1] = ii+1;
25      }
26
27
28    double A[nnodes][nnodes];
29    for (unsigned int ii = 0; ii < nnodes; ++ii)
30      for (unsigned int jj = 0; jj < nnodes; ++jj)
31        A[ii][jj] = 0.0;
32
33    for (unsigned int iel = 0; iel < nels; ++iel)
34      {
```

Implementation in C++

```
30        for (unsigned int jj = 0; jj < nnodes; ++jj)
31          A[ii][jj] = 0.0;
32
33      for (unsigned int iel = 0; iel < nels; ++iel)
34        {
35
36          double mloc[2][2];
37          for (unsigned int ii = 0; ii < 2; ++ii)
38            for (unsigned int jj = 0; jj < 2; ++jj)
39              mloc[ii][jj] = 0.0;
40
41          for (unsigned int inode = 0; inode < 2; ++inode)
42            {
43              double igrad = (inode == 0 ? 1.0 / h : -1.0 / h);
44              for (unsigned int jnode = 0; jnode < 2; ++jnode)
45                {
46                  double jgrad =  (jnode == 0 ? 1.0 / h : -1.0 / h);
47                  mloc[inode][jnode] = igrad * jgrad * h;
48                  A[elements[iel][inode]][elements[iel][jnode]] +=
49                    mloc[inode][jnode];
50                }
51            }
52        }
53
54      double f[nnodes];
55      for (unsigned int ii = 0; ii < nnodes; ++ii)
56        f[ii] = 0.0;
57
58      for (unsigned int iel = 0; iel < nels; ++iel)
59        {
60          double vloc[2];
61          for (unsigned int ii = 0; ii < 2; ++ii)
62            vloc[ii] = 0.0;
63
64          for (unsigned int inode = 0; inode < 2; ++inode)
65            {
```

Implementation in C++

```
60          double vloc[2];
61          for (unsigned int ii = 0; ii < 2; ++ii)
62            vloc[ii] = 0.0;
63
64          for (unsigned int inode = 0; inode < 2; ++inode)
65            {
66              vloc[inode] = h / 2.0;
67              f[elements[iel][inode]] += vloc[inode];
68            }
69        }
70
71      f[0] = 0;
72      f[nnodes - 1] = 0;
73
74      A[0][0] = 1.0;
75      for (unsigned int ii = 1; ii < nnodes; ++ii)
76        A[0][ii] = 0.0;
77
78      A[nnodes-1][nnodes-1] = 1.0;
79      for (unsigned int ii = 0; ii < nnodes-1; ++ii)
80        A[nnodes-1][ii] = 0.0;
81
82      double uh[nnodes];
83      for (unsigned int ii = 0; ii < nnodes; ++ii)
84        uh[ii] = 0.0;
85
86      double uh_new = 0;
87      double incrnorm = 0;
88      for (unsigned int ii = 0; ii < maxit; ++ii)
89        {
90          incrnorm = 0;
91          for (unsigned int jj = 0; jj < nnodes; ++jj)
92            {
93              double res = f[jj];
94
95              for (unsigned int kk = 0; kk < nnodes; ++kk)
```

Implementation in C++

```
90          incrnorm = 0;
91          for (unsigned int jj = 0; jj < nnodes; ++jj)
92            {
93              double res = f[jj];
94
95              for (unsigned int kk = 0; kk < nnodes; ++kk)
96                if (kk != jj)
97                  res -= A[jj][kk] * uh[kk];
98
99              uh_new = res / A[jj][jj];
100             incrnorm = std::abs (uh_new - uh[jj]) > incrnorm ?
101               std::abs (uh_new - uh[jj]) :
102               incrnorm;
103             uh[jj] = uh_new;
104           }
```

Compile and run

```
$ g++ -std=gnu++11 fem1d.cpp -o fem1d
$ ./fem1d >> res.txt
$ octave -q
>> load res.txt
>> plot (res)
```

Compile with a Makefile

```
fem1d : fem1d.cpp
        $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $<
        $(CXX) $(LDFLAGS) $@.o -o $@ $(LIBS)

.PHONY: clean distclean

clean :
        $(RM) fem1d.o

distclean : clean
        $(RM) fem1d
```

Refactored Implementation in C++

file v0.2/fem1d.cpp

```
1   #include "fem1d.h"
2
3   int main ()
4   {
5
6     mesh m (a, b, nnodes);
7
8     double A[nnodes][nnodes];
9     std::fill (&(A[0][0]), &(A[nnodes-1][nnodes]), 0.0);
10
11    for (unsigned int iel = 0; iel < m.nels; ++iel)
12      {
13
14        double mloc[2][2];
15        std::fill (&(mloc[0][0]), &(mloc[1][2]), 0.0);
16
17        for (unsigned int inode = 0; inode < 2; ++inode)
18          {
19            double igrad = (inode == 0 ? 1.0 / m.h : -1.0 / m.h);
20            for (unsigned int jnode = 0; jnode < 2; ++jnode)
21              {
22                double jgrad =  (jnode == 0 ? 1.0 / m.h : -1.0 / m.h);
23                mloc[inode][jnode] = igrad * jgrad * m.h;
24                A[m.elements[iel][inode]][m.elements[iel][jnode]] +=
25                  mloc[inode][jnode];
26              }
27          }
28      }
29
30    double f[nnodes];
31    std::fill (f, f + nnodes, 0.0);
32
33    for (unsigned int iel = 0; iel < m.nels; ++iel)
34      {
35        double vloc[2];
```

Refactored Implementation in C++

file v0.2/fem1d.cpp

```
30      double f[nnodes];
31      std::fill (f, f + nnodes, 0.0);
32
33      for (unsigned int iel = 0; iel < m.nels; ++iel)
34        {
35          double vloc[2];
36          std::fill (vloc, vloc + 2, 0.0);
37
38          for (unsigned int inode = 0; inode < 2; ++inode)
39            {
40              vloc[inode] = m.h / 2.0;
41              f[m.elements[iel][inode]] += vloc[inode];
42            }
43        }
44
45      f[0] = 0;
46      f[nnodes - 1] = 0;
47
48      A[0][0] = 1.0;
49      std::fill (&(A[0][1]), &(A[0][nnodes]), 0.0);
50
51      A[nnodes-1][nnodes-1] = 1.0;
52      std::fill (&(A[nnodes-1][0]), &(A[nnodes-1][nnodes-1]), 0.0);
53
54      double uh[nnodes];
55      gauss_seidel (A, f, uh);
56
57      for (unsigned int ii = 0; ii < nnodes; ++ii)
58        std::cout << uh[ii] << std:: endl;
59
60      return 0;
61    };
```

Refactored Implementation in C++

file `v0.2/fem1d.h`

```
1   #ifndef HAVE_FEM1D_H
2   #define HAVE_FEM1D_H
3
4   #include <array>
5   #include <algorithm>
6   #include <cmath>
7   #include <iostream>
8
9   #include "config.h"
10  #include "mesh.h"
11  #include "gauss_seidel.h"
12
13  #endif
```

Refactored Implementation in C++

file `v0.2/config.h`

```
1  #ifndef HAVE_CONFIG_H
2  #define HAVE_CONFIG_H
3
4  constexpr double a = 0;
5  constexpr double b = 1;
6  constexpr unsigned int nnodes = 50;
7
8  #endif
```

Refactored Implementation in C++

file v0.2/mesh.h

```
1   #ifndef HAVE_MESH_H
2   #define HAVE_MESH_H
3
4   class mesh
5   {
6
7   public:
8
9     const unsigned int nnodes;
10    const double L;
11    const unsigned int nels;
12    const double h;
13
14    double *nodes;
15    unsigned int (*elements)[2];
16
17    mesh
18    (const double a, const double b, const unsigned int nnodes_);
19
20    ~mesh ()
21    {
22      delete [] nodes;
23      delete [] elements;
24    }
25
26  };
27
28  #endif
```

Refactored Implementation in C++

file v0.2/mesh.cpp

```
1   #include "mesh.h"
2
3   mesh::mesh
4   (const double a, const double b, const unsigned int nnodes_)
5     : L (b - a), nnodes (nnodes_),
6       nels (nnodes - 1), h (L / double (nels))
7     {
8
9       nodes = new double [nnodes];
10      for (unsigned int ii = 0; ii < nnodes; ++ii)
11        nodes[ii] = static_cast<double>(ii) * h + a;
12
13      elements = new unsigned int [nels][2];
14      for (unsigned int ii = 0; ii < nels; ++ii)
15        {
16          elements[ii][0] = ii;
17          elements[ii][1] = ii+1;
18        }
19    }
```

Refactored Implementation in C++

file v0.2/gauss_seidel.h

```
1  #ifndef HAVE_GAUSS_SEIDEL_H
2  #define HAVE_GAUSS_SEIDEL_H
3
4  #include "config.h"
5
6  void
7  gauss_seidel
8  (const double A[][nnodes],
9   const double f[],
10  double uh[],
11  const unsigned int maxit = 10000,
12  const double tol = 1.0e-15);
13
14
15 #endif
```

Refactored Implementation in C++

file v0.2/gauss_seidel.cpp

```
1  #include "gauss_seidel.h"
2  #include <cmath>
3
4  void
5  gauss_seidel
6  (const double A[][nnodes],
7   const double f[],
8   double uh[],
9   const unsigned int maxit,
10  const double tol)
11 {
12   double uh_new = 0;
13   double incrnorm = 0;
14   for (unsigned int ii = 0; ii < maxit; ++ii)
15     {
16       incrnorm = 0;
17       for (unsigned int jj = 0; jj < nnodes; ++jj)
18         {
19           double res = f[jj];
20
21           for (unsigned int kk = 0; kk < nnodes; ++kk)
22             if (kk != jj)
23               res -= A[jj][kk] * uh[kk];
24
25           uh_new = res / A[jj][jj];
26           incrnorm = std::abs (uh_new - uh[jj]) > incrnorm ?
27             std::abs (uh_new - uh[jj]) :
28             incrnorm;
29           uh[jj] = uh_new;
30         }
31       if (incrnorm < tol)
32         break;
33     }
34 }
```

Compile and run

```
g++ -std=c++11 -c fem1d.cpp -I.
g++ -std=c++11 -c mesh.cpp -I.
g++ -std=c++11 -c gauss_seidel.cpp -I.
g++ -o fem1d fem1d.o gauss_seidel.o mesh.o
rm fem1d.o gauss_seidel.o mesh.o
```

Compile with a Makefile

```
OBJS = fem1d.o gauss_seidel.o mesh.o
HEADERS = config.h

fem1d : $(OBJS)
        $(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)

$(OBJS) : %.o : %.cpp %.h $(HEADERS)
        $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $<

.PHONY: clean distclean

clean :
        $(RM) $(OBJS)

distclean : clean
        $(RM) fem1d
```

Refactored Implementation in C++ (templates + STL)

file v0.3/fem1d.cpp

```
1   #include "fem1d.h"
2
3   int main ()
4   {
5
6     mesh<nnodes> m (a, b);
7
8     std::array<std::array<double, nnodes>, nnodes> A;
9     for (unsigned int ii = 0; ii < nnodes; ++ii)
10      A[ii].fill (0.0);
11
12    for (unsigned int iel = 0; iel < m.nels; ++iel)
13      {
14        std::array<std::array<double, 2>, 2> mloc;
15        for (unsigned int ii = 0; ii < 2; ++ii)
16          mloc[ii].fill (0.0);
17
18        for (unsigned int inode = 0; inode < 2; ++inode)
19          {
20            double igrad = (inode == 0 ? 1.0 / m.h : -1.0 / m.h);
21            for (unsigned int jnode = 0; jnode < 2; ++jnode)
22              {
23                double jgrad =  (jnode == 0 ? 1.0 / m.h : -1.0 / m.h);
24                mloc[inode][jnode] = igrad * jgrad * m.h;
25                A[m.elements[iel][inode]][m.elements[iel][jnode]] +=
26                  mloc[inode][jnode];
27              }
28          }
29      }
30
31    std::array<double, nnodes> f;
32    f.fill (0.0);
33    for (unsigned int iel = 0; iel < m.nels; ++iel)
34      {
35        std::array<double, 2> vloc;
```

Refactored Implementation in C++ (templates + STL)

file `v0.3/fem1d.cpp`

```
30
31      std::array<double, nnodes> f;
32      f.fill (0.0);
33      for (unsigned int iel = 0; iel < m.nels; ++iel)
34        {
35          std::array<double, 2> vloc;
36          vloc.fill (0.0);
37
38          for (unsigned int inode = 0; inode < 2; ++inode)
39            {
40              vloc[inode] = m.h / 2.0;
41              f[m.elements[iel][inode]] += vloc[inode];
42            }
43        }
44
45      f[0] = 0;
46      f.back () = 0;
47
48      A[0][0] = 1.0;
49      std::fill (&(A[0][1]), A[0].end (), 0.0);
50
51      A.back ().back () = 1.0;
52      std::fill (A.back ().begin (), A.back ().end () - 1, 0.0);
53
54      std::array<double, nnodes> uh (f);
55      gauss_seidel<nnodes> (A, f, uh);
56
57      for (unsigned int ii = 0; ii < nnodes; ++ii)
58        std::cout << uh[ii] << std:: endl;
59
60      return 0;
61    };
```

Refactored Implementation in C++ (templates + STL)

file `v0.3/fem1d.h`

```
1   #ifndef HAVE_FEM1D_H
2   #define HAVE_FEM1D_H
3
4   #include <array>
5   #include <algorithm>
6   #include <cmath>
7   #include <iostream>
8
9   #include "config.h"
10  #include "mesh.h"
11  #include "gauss_seidel.h"
12
13  #endif
```

Refactored Implementation in C++ (templates + STL)

file `v0.3/config.h`

```
1  constexpr double a = 0;
2  constexpr double b = 1;
3  constexpr unsigned int nnodes = 50;
```

Refactored Implementation in C++ (templates + STL)

file v0.3/mesh.h

```
1  #ifndef HAVE_MESH_H
2  #define HAVE_MESH_H
3
4  template<unsigned int nnodes>
5  class mesh
6  {
7
8  public:
9
10   const double L;
11   const unsigned int nels;
12   const double h;
13
14   std::array<double, nnodes> nodes;
15   std::array<std::array<unsigned int, 2>, nnodes - 1> elements;
16
17   mesh (const double a, const double b) : L (b - a),
18                                           nels (nnodes - 1),
19                                           h (L / double (nels))
20   {
21     for (unsigned int ii = 0; ii < nnodes; ++ii)
22       nodes[ii] = static_cast<double>(ii) * h + a;
23
24     for (unsigned int ii = 0; ii < nels; ++ii)
25       {
26         elements[ii][0] = ii;
27         elements[ii][1] = ii+1;
28       }
29   }
30
31 };
32
33 #endif
```

Refactored Implementation in C++ (templates + STL)

file v0.3/gauss_seidel.h

```
1  #ifndef HAVE_GAUSS_SEIDEL_H
2  #define HAVE_GAUSS_SEIDEL_H
3
4  template<unsigned int nnodes>
5  void
6  gauss_seidel (const std::array<std::array<double, nnodes>, nnodes> &A,
7                const std::array<double, nnodes> &f,
8                std::array<double, nnodes> &uh,
9                const unsigned int maxit = 10000,
10               const double tol = 1.0e-9)
11 {
12   double uh_new = 0;
13   double incrnorm = 0;
14   for (unsigned int ii = 0; ii < maxit; ++ii)
15     {
16       incrnorm = 0;
17       for (unsigned int jj = 0; jj < nnodes; ++jj)
18         {
19           double res = f[jj];
20
21           for (unsigned int kk = 0; kk < nnodes; ++kk)
22             if (kk != jj)
23               res -= A[jj][kk] * uh[kk];
24
25           uh_new = res / A[jj][jj];
26           incrnorm = std::abs (uh_new - uh[jj]) > incrnorm ?
27             std::abs (uh_new - uh[jj]) :
28             incrnorm;
29           uh[jj] = uh_new;
30         }
31       if (incrnorm < tol)
32         break;
33     }
34 }
```

Compile and run

```
g++ -std=c++11 -c fem1d.cpp -I.
g++ -o fem1d fem1d.o
rm fem1d.o
```

Compile with a Makefile

```
OBJS = fem1d.o
HEADERS = fem1d.h mesh.h gauss_seidel.h config.h

fem1d : fem1d.cpp $(HEADERS)
        $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $<
        $(CXX) $(LDFLAGS) $@.o -o $@ $(LIBS)

.PHONY: clean distclean

clean :
        $(RM) $(OBJS)

distclean : clean
        $(RM) fem1d
```